МИНОБРНАУКИ РОССИИ
САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ
ЭЛЕКТРОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ
«ЛЭТИ» ИМ. В.И. УЛЬЯНОВА (ЛЕНИНА)
Кафедра МО ЭВМ

ОТЧЕТ

по лабораторной работе №2

по дисциплине «Алгоритмы и структуры данных»

Тема: Иерархические списки

Студент гр. 8304 _____ Карабанов Р.Е

Преподаватель _____ Фирсов М.А

Санкт-Петербург

2019

Цель работы.

Ознакомиться с основными понятиями и структурой иерархических списков.

Задание.

Вариант № 7.

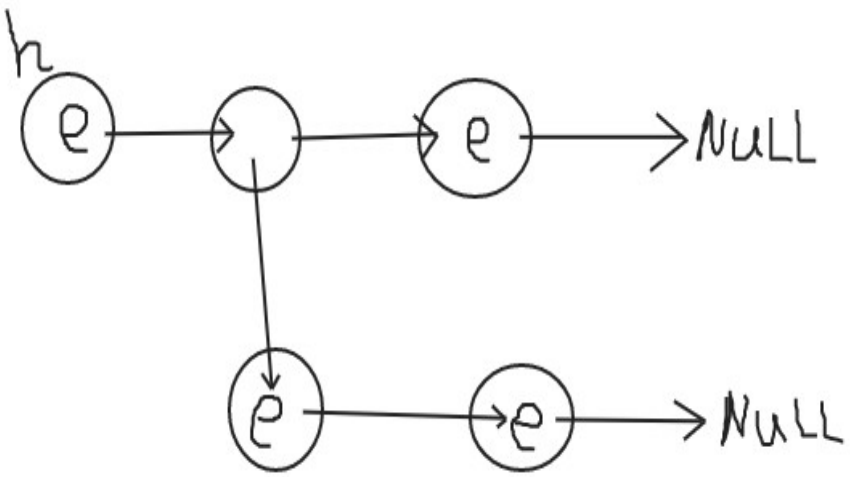
удалить из иерархического списка все вхождения заданного элемента (атома) x;

Описание функций краткое описание алгоритма.

В начале работы пользователю предлагается выбрать способ ввода иерархического списка — с консоли или из файла. Далее проводится проверка введённой строки на корректность с помощью функции unsined int CheckStr(std::string s1) функция принимает строку — список и проверяет её на корректность, возвращает 0 в случае, если строка некорректна. Затем в случае корректной строки вызывается функция list* ReadList(std::string s1, unsigned int* i, list* parent) — это рекурсивная функция создания списка из введённой строки, принимает строку, указатель, по которому отслеживается символ в строке и указатель на элемент, который является отцовским для создаваемого, возвращает указатель на «голову» иерархического списка. Далее пользователем вводится символ который необходимо удалить. Затем вызывается функция list* CheckList(list* head, char ellemfordel) — это рекурсивная функция просмотра списка на наличия в нём элемента, подлежащего удалению, возвращает указатель на проверенный список. Для удаления элемента вызывается функция list* DeleteEllem(list** head, list** cur) — возвращает укащател на «голову» списка, в котором удалён элемент. Далее на консоль выводится список после операций с помощью функции void

PrintList(list*head). Затем, чтобы освободить выделенную пол список память вызывается функция destroy(list** head) в которой происходит удаление списка.

Представление иерархического списка в памяти.



Тестирование.

```
g++ ./Source/ADS2.cpp -Wall -Werror -o ADS_lab2
> Choose your input
> 0 - from console
> 1 - from file
> Any other to Exit
> 1
> FilePath: ./Tests/test.txt
Reading from file:

aba
test #1 "aba"
> Enter ellement for delete : a
> list after delete : "b"
a(b(a))a
test #2 "a(b(a))a"
> Enter ellement for delete : a
> list after delete : "(b)"
a((a)b)a
test #3 "a((a)b)a"
> Enter ellement for delete : b
> list after delete : "a((a))a"
```

Ниже представлена таблица № 1, в которой представлены некоторые примеры работы программы.

Таблица №1 примеры работы программы.

Строка	Символ для удаления	Вывод
aba	f	aba
)(	-	Wrong List
abdefd(d)	d	abef

Выводы.

В ходе лабораторной работы были изучены основные понятия и структура иерархических списков. Получены навыки реализации иерархических списков.

Исходный код.

```
#include <iostream>
#include <string>
#include <fstream>

typedef char base;

struct list {
	base ellem;
	list* next;
	list* parent;
	list* child;
};

/* Проверка строки, введённой пользователем на корректность */
int CheckStr(std::string s1) {
	bool AlphaInScope = false;
	int i = 0;
	for (; i < s1.length(); i++) {
		if (s1.at(i) == '(') {
			while (i < s1.length()) {
				if (s1.at(i) == ' ') {
					return 0;
				}
				else if (s1.at(i) == '(' && !AlphaInScope) {
					return 0;
				}
				else {
					AlphaInScope = true;
				}
				i++;
			}
		}
		else if (s1.at(i) == ' ' || s1.at(i) == ')') {
			return 0;
		}
	}
	return i;
```

```
}

/* Рекурсивное создание иерархического списка,
использование unsigned int* i - для избежания глобальной переменной */
list* ReadList(std::string s1, unsigned int* i, list* parent) {
	list* head = new list;
	list* cur = head;
	while (*i < s1.length()) {
		if (s1.at(*i) == '(') {
			(*i) += 1;
			cur->child = ReadList(s1, i, cur);
			cur->ellem = '1';
		}
		else if (s1.at(*i) == ')') {
			(*i)++;
			cur->next = nullptr;
			return head;
		}
		else {
			cur->ellem = s1.at(*i);
			cur->child = nullptr;
			cur->parent = parent;
			(*i)++;
		}
		if (*i < s1.length() && s1.at(*i) != ')') {
			cur->next = new list;
			cur = cur->next;
		}
	}
	cur->next = nullptr;
	return head;
}

/* Функция удаления эллемента из списка */
list* DeleteEllem(list** head, list** cur) {
	list* ptr;
	if (*cur == *head) {
		*head = (*head)->next;
		delete(*cur);
```

```
		(*cur) = nullptr;
		*cur = *head;
	}
	else {
		ptr = *head;
		while (ptr->next != *cur) {
			ptr = ptr->next;
		}
		ptr->next = (*cur)->next;
		delete(*cur);
		*cur = ptr;
	}
	return *head;
}

/* Функция поиска и удаления эллемента списка, удовлетворяющего условиям*/
list* CheckList(list* head, char ellemfordel) {
	list* cur = head;
	while (cur != nullptr) {
		if (cur->ellem == ellemfordel) {
			if (cur == head) {
				head = DeleteEllem(&head, &cur);
				continue;
			}
			else {
				head = DeleteEllem(&head, &cur);
			}
		}
		else if (cur->child != nullptr) {
			cur->child = CheckList(cur->child, ellemfordel);
			if (cur->child == nullptr) {
				if (cur == head) {
					head = DeleteEllem(&head, &cur);
					continue;
				}
				else {
					head = DeleteEllem(&head, &cur);
				}
			}
```

```
                }
                cur = cur->next;
        }
        return head;
}


void PrintList(list* head) {
        list* cur = head;
        while (cur != nullptr) {
                if (cur->child != nullptr) {
                        std::cout << '(';
                        PrintList(cur->child);
                        std::cout << ')';
                }
                else {
                        std::cout << cur->ellem;
                }
                cur = cur->next;
        }
}


void destroy(list** head) {
        list* cur = *head;
        while (*head != nullptr) {
                if ((*head)->child != nullptr) {
                        destroy(&(cur->child));
                }
                else {
                        cur = (*head)->next;
                        delete(*head);
                        *head = cur;
                }
        }
}

void execute(std::string listStr) {
        unsigned int i = 0;
```

```
	list* head = ReadList(listStr, &i, nullptr);
	std::cout << "> Enter ellement for delete : ";
	char ellemfordel;
	std::cin >> ellemfordel;
	head = CheckList(head, ellemfordel);
	std::cout << "> list after delete : \"";
	PrintList(head);
	std::cout << "\"" << std::endl;
	destroy(&head);
}

void ReadFromFile(std::string filename)
{
	std::ifstream file(filename);
	if (file.is_open())
	{
		std::cout << "Reading from file:" << "\n\n";
		int count = 0;
		std::string listStr;
		while (std::getline(file, listStr))
		{
			count++;
			std::cout << listStr << std::endl;
			if (!CheckStr(listStr)) {
				std::cout << "> Wrong List" << std::endl;
				continue;
			}
			std::cout << "test #" << count << " \"" + listStr + "\"" << "\n";
			execute(listStr);
		}
	}
	else
	{
		std::cout << "File not opened" << "\n";
	}
}
```

```
int main(int argc, char* argv[])
{
	std::cout << "> Choose your input" << std::endl;
	std::cout << "> 0 - from console" << std::endl;
	std::cout << "> 1 - from file" << std::endl;
	std::cout << "> Any other to Exit" << std::endl;
	std::cout << "> ";
	char command = '3';
	std::cin.get(command);
	std::cin.get();

	switch (command) {
	case '0': {
		std::string input;
		std::cout << "> Enter the List without spaces: ";
		std::getline(std::cin, input);
		if (!CheckStr(input)) {
			std::cout << "> Wrong List" << std::endl;
			return 1;
		}
		execute(input);
		break;
	}
	case '1': {
		std::cout << "> FilePath: ";
		std::string filePath;
		if (argc > 1) {
			filePath = argv[1];
			std::cout << filePath << std::endl;
		}
		else
			std::cin >> filePath;
		ReadFromFile(filePath);
		break;

	}
	case '3':
	default:
		std::cout << "> Error command \n> end\n";
```

```
        }
        return 0;
}
```